FPGA IMPLEMENTATION OF TRANSMITTER OF THE MULTI SYMBOL ENCAPSULATED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING

A PROJECT REPORT

Submitted in fulfillment of the requirements for the award of the degree of

Master of Technology in
the Specialization of VLSI and Embedded Systems
In Electronics and Communication Engineering

By

T ASHOKKUMAR (211EC2090)

Under the guidance of

POONAM SINGH
Associate Professor

Department of Electronics & Communication Engineering
National Institute of Technology
Rourkela
2013

FPGA IMPLEMENTATION OF TRANSMITTER OF THE MULTI SYMBOL ENCAPSULATED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING

A PROJECT REPORT

Submitted in fulfillment of the requirements for the award of the degree of

Master of Technology in
the Specialization of VLSI and Embedded Systems
In Electronics and Communication Engineering

By

T ASHOKKUMAR (211EC2090)

Under the guidance of

POONAM SINGH
Associate Professor

Department of Electronics & Communication Engineering
National Institute of Technology
Rourkela
2013

Department of Electronics & Communication Engineering
National Institute of Technology

CERTIFICATE AUTHENTICITY

This is to cetify that the project entitled **"FPGA implementation of Transmitter of Multi Symbol Encapsulated Orthogonal Frequency Division Multiplexing"** being submitted by Mr.T.Ashokkumar (211EC2090) in fullment of the requirements for the award of the degree of Master of Technology in Electronics & Communication Engineering Department of National Institute of Technology, Roukela is an authentic work carried out by him under my supervision and guidance.

Mrs. Poonam Singh
Associate Professor
Department of ECE
NIT ROURKELA

ACKNOWLEDGEMENT

I take this opportunity to express my heart-felt gratitude for everyone without whose support and encouragement this project would not have been successful. I would like to thank my project guide, Mrs. Poonam Singh for her constant support and advice throughout the year. I would like to express my thanks to her for helping me out of tough situations and being a constant support during programming presentations etc…

I would also like to thank all professors and PhD students who have been a constant and patient help with VHDL and successfully implementing my project.

Lastly I convey my thanks all my dear friends those who helped me in several ways in completion of this project successfully.

T ASHOKKUMAR

211EC2090

DECLARATION

I hereby declare that the project entitled **" FPFA implementation of Transmitter of Multi Symbol Encapsulated Orthogonal Frequency Division Multiplexing"** is bonafide work carried by me under the supervision of **Mrs .POONAM SINGH** in partial fulfillment for the award of degree of Master of Technology in ELECTRONICS AND COMMUNICATION ENGINEERING.

T ASHOKKUMAR

211EC2090

Contents

Abstract:

MSE-OFDM is a new multi carrier system, in which one cyclic prefix used for multiple OFDM symbols. The original motivation for the MSE-OFDM proposal is to reduce the redundancy caused by the cyclic prefix. In this paper the design implementation of MSE-OFDM transmitter on a field programmable gate array has been presented. The implementation is made on FPGA since it allows flexibility in design and also it can achieve higher computing speed than digital signal processors . IDFT block in MSE-OFDM transmitter is implemented by IFFT to enhance the speed. This architecture uses radix-2 algorithm for 8 point IFFT and radix-4 algorithm for 16-point IFFT. Here advantage is that IFFT algorithms use only adders, subtractors and shifter only thus minimizing the hardware requirement compared to other architectures. Here implementation of 'binary source using pseudo random binary generator' and 'M-QAM' has also been presented. Here the signed numbers are represented using 12-bit signed binary 6.6 fixed point representation so that to get more accurate results even for decimal values. The The proposed design is mapped on to Xilinx virtex-5 xc5vlx30-2ff676 based device. XST synthesis tool has been used to synthesize the model. The result shows that maximum frequency of operation is in range of 600- 700 MHz . the system design is optimized in terms of speed,power and area.

List of figures

List of tables

List of abbreviations

OFDM - Orthogonal Frequency Division Multiplexing

MSE-OFDM - Multi Symbo Encapsulated Orthogonal Frequency Division Multiplexing

QAM - Qadrature Amplitude Modulation

CIR – Channel Impulse Response

CP – Cyclic Prefix

VHDL – Very high speed Hardware Description Language

LFSR – Linear Feedback Shift Register

IFFT – Inverse Fast fourier Transform

NEDA – New Distributed Arithmetic

DA – Destributed arithmetic

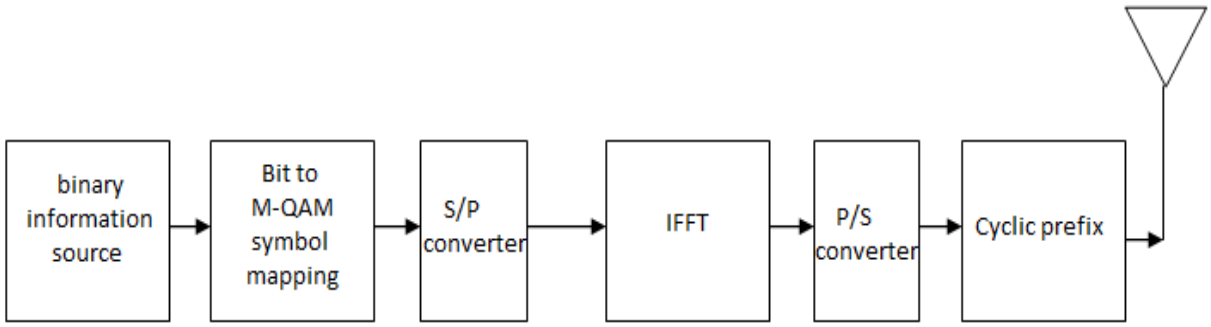
Chapter 1

Introduction

Comparisom between OFDM and MSE-OFDM (Transmitter):

OFDM transmitter:

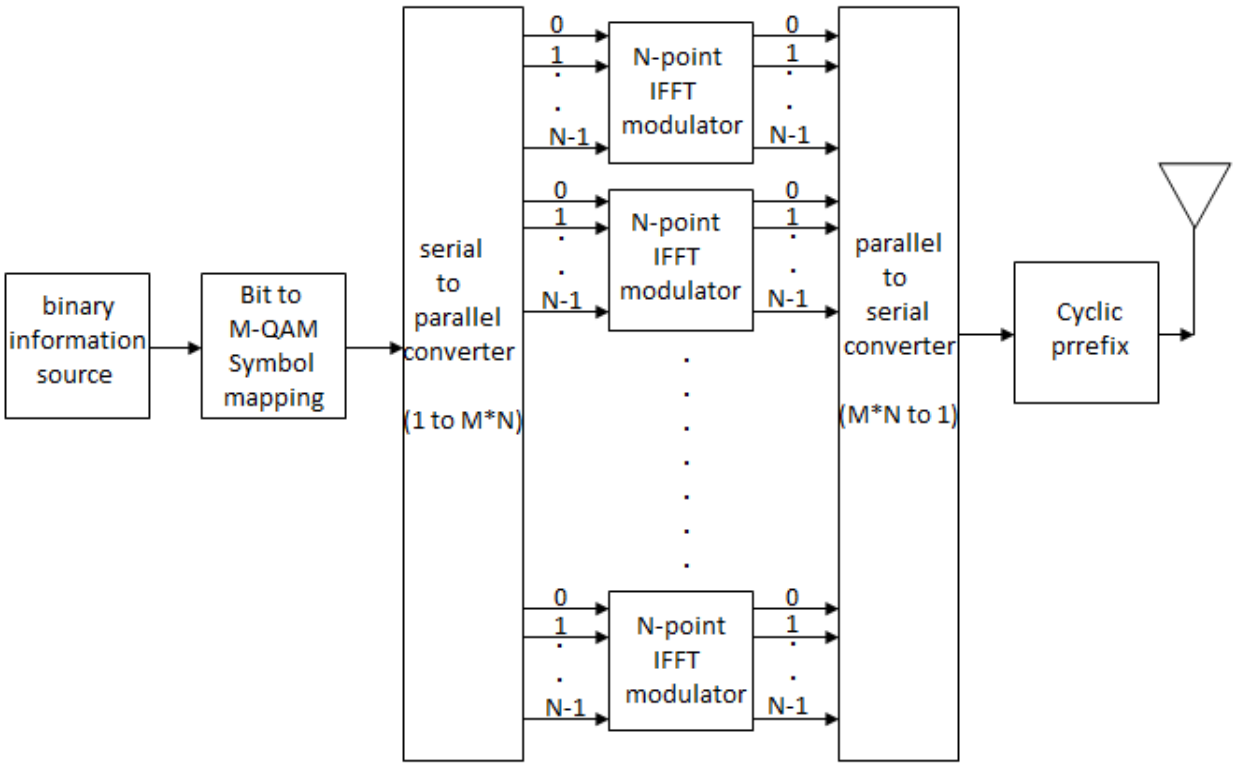
Single OFDM transmitter transmits one OFDM symbol at a time and cyclic prefix is added to this symbol. The block diagram for OFDM transmitter is shown in following figure



The CP, which is a cyclic extension of the inverse discrete Fourier transformation (IDFT) output, has to be at least as long as the channel impulse response in order to avoid the inter symbol interference. Therefore, redundancy is unavoidably introduced in conventional OFDM systems. This restricts the achievable bandwidth efficiency especially for channels with very long channel impulse response .

so a technique of grouping multiple single-carrier symbols into a frame with a CP has been proposed. This approach essentially emulates the time-domain signal structure of the conventional OFDM system by providing a cyclic data frame longer than the channel-response time. In this paper, we propose an MSE-OFDM scheme that uses a different type of CP: Instead of using one CP for each OFDM symbol, a number of OFDM symbols are grouped together as a frame and protected by a single CP. The block diagram for this is shown in figure. In the diagram as we seen multiple IFFT block present each of which represent one OFDM symbol.

MSE-OFDM transmitter:



IDFT is the main block in MSE-OFDM transmitter. N-point IDFT requires arithmetic operations of the order of $O(N^2)$ that is N^2 multiplications and N (N-1) additions. Where as for IFFT requires arithmetic operations of the order of O(NlogN) that is NlogN additions and (N/2)logN multiplications. So IFFT is prefered here. DA has become one of the most efficient tools in VLSI implementation of digital signal processing (DSP) architectures. It efficiently computes inner products of vectors, which is a key requirement in many DSP systems. One of the key computational blocks in DSP is MAC, which is implemented by a standard adder unit and a multiplier. Using DA, MAC unit can be implemented by pre computing all possible products and using a ROM to store them. The con of using DA is in its exponential increase of the size of ROM with increase in internal precision and number of inputs. An approach to overcome this drawback is by distributing the coefficients to inputs. One of such examples is new distributed arithmetic (NEDA). Here this algorithm does not requires any ROM or multiplier units thus it is an efficient one as it is being implemented using only adders/subtractors and shifters.

Literature survey:

[1] Describes how to improve the OFDM system interms of bandwidth efficiency using MSE-OFDM . In this paper how to over come the redundancy due to CP using new transmission technique has been proposed.

[2] In this paper comparison of OFDM and MSE-OFDM has been proposed. In this paper how multiple symbols are transmitted using MSE-OFDM and single OFDM symbol transmitted using OFDM system has been proposed.

[9] In this paper how to implement the IDFT using IFFT algorithms has been described. Here IFFT is implementd using adder/ subtractors and shifter only using a new technique called new distributed algorithm (NEDA). Necessity of ROM and multiplier using Distributed algorithm has been removed in this NEDA technique .

[11] In this paper how to generate random binary number using pseudo random binary sequence has been presented. Here pseudo random binary sequence is designed using LFSR .

[12] In this paper implementation of LFSR using linear function XOR and feed back of of some bit position(taps) using this linear function XOR as input and how register states will vary for every clock cycles has been presented.

[16] In this paper signed binary fixed point representation is described which is necessary to get most accurate results even for decimal values obtained at the IFFT output.

Chapter 2

VHDL- The Language of Hardware

VHDL code consists of 3 parts

Library declarations

Entity declarations

Architecture declarations

Libraries/packages declarations :

```
Library library_name;
Use library_name.package_name.package_parts;
```

'std' , 'work' and 'ieee' are the important libraries we frequently used in VHDL code. Their declarations are as follows.

```
Library std;
Library ieee;
Use ieee.std_logic_1164.all;
Library work;
Use work.all;
```

The libraries std and work shown above are made visible by default, so there is no need to declare them. Only ieee library must be explicitly written. The purpose of three libraries/packages mentioned above is following.

Package standard of library std : it defines BIT, BOOLEAN, INTEGER, and REAL data types.

Std is resource library(datatypes, text i/o etc) and the work library is where we save our design (the .vhd file, and all files created by the compiler, simulator, etc).

The std_logic_1164 package of the ieee library specifies a multi-level logic system;

Indeed the ieee library contains several packages including the following.

Std_logic_1164: specifies the **std_logic** (8-levels) and **std_ulogic**(9-levels) multi-valued logic system.

Std_logic_arith: it specifies the signed and unsigned data types and related arithmetic and comparison operations. It also contains several data conversion functions, which allows one type to be converted into another. Eg:conv_integer(p):

Entity declarations:

In this entity inputs and required output are defined .

Eg: entity entity_name is

Port(x:in std_logic;

Y: out std_logic);

End entity_name;

Architecture declaration:

This code is written in this architecture . this code may be concurrent or sequential. In concurrent code statements will be executed parallelly and in sequential code statements will be executed sequentially.

Data types:

two types of datatypes are present.

Pre defined datatypes: in VHDL some Pre defined datatypes are presented.

They are for, in , generate,signal, variable etc.

user defined datatypes:

VHDL allows the user to define own data types in following two ways using TYPE . they are

a) user defined integer types:

ex:TYPE integer is RANGE -32 to 32;

b)user defined enumerated types:

ex: TYPE vector is array (0 to 31) of signed(11 downto 0));

TYPE state is (stateA,stateB,stateC,stateD) ;

Arrays:

Arrays are collections of objects of the same type. They can be one-dimensional(1D), two-dimensional(2D), or one-dimensional-by-one-dimensional(1D*1D). Predefined datatypes include only the scalar(single bit) and vector (one dimensional array of bits) ex:

Scalars: BIT, STD_LOGIC etc.

Vectors: BIT_VECTOR, STD_LOGIC_VECTOR,SIGNED etc.

There are no predefined 2D or 1D*1D arrays. So to specify them the new TYPE must first be defined. then the new SIGNAL, VARIABLE or CONSTANT can be declared using that data type.

To specify a new array type:

Type type_name is array(specification) of data_type;

To make use of new array type:

SIGNAL signal_name : type_name [:= initial_value];

Ex: type state is (stateA,stateB,stateC,stateD,stateE)

TYPE vector1 is array (0 to 31) of STD_LOGIC; 1D array

TYPE vector2 is array (0 to 31) of STD_LOGIC_VECTOR (11 downto 0)); 1D*1D array

TYPE vector3 is array (0 to 31, 11 downto 0) of STD_LOGIC; 2D array

To use above type signal declared as following

Signal prstate,nxstate : vector1;

Signal x : vector1;

Signal y : vector2;

Type declarations are not allowed inside entity. so To specify the in/out ports as arrays of vectors inside entity user defined datatypes should be declared in a PACKAGE which is visible to the whole design including entity

Example :

```
------------------------------package------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE my_pkg is
Type vector is array(0 to 15) of signed(11 downto 0).
End my_pkg;
------------------------------main code------------------------------

LIBRARY ieee;
```

```
USE   ieee.std_logic_1164.all;
USE work.my_pkg.all;    ------user defined package
Entity   fft   is
Port(x: in vector;
      Y: out vector);
End  fft;
```

Signed and unsigned datatypes:

Their Syntax is similar to std_logic_vector

```
Signal  x: signed(7 downto 0);
Signal  y: unsigned(0 to 3);
```

Unsigned value always represents positive value.

Eg:"0101" represents decimal 5.

"1101" represents decimal 13.

Signed value can be positive or negative(in two's complement format) depends on MSB bit. If MSB is '0' number is positive, if it is '1' it is negative.

Data conversion:

VHDL does not allow direct operations between data of different types. So it is necessary to convert data from one type to another. Several data conversion functions can be found in the std_logic_arith package of the ieee library. They are

Conv_integer(p): converts a parameter 'p' of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an INTEGER value.

Conv_std_logic_vector(p,b): converts a parameter ' p' of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size 'b' bits.

Ex:conv_std_logic_vector(3,4) gives "0011" which of std_logic_vector value with sized '4' bits.

Operators :

a) Assignment operators:

<= used to assign a value to a signal.

:= used to assign a value to a VARIABLE, CONSTANT, or GENERIC. Used also for establishing initial values.

'=>' used for establishing initial values.

Eg signal x: std_logic;

Eg variable y :std_logic_vector(3 downto 0);

X <= '1' ;

Y := "1001";

b)Logical operators: they are used to perform logical operations if the data must be of type BIT, STD_LOGIC, STD_ULOGIC or their corresponding vectors (eg: BIT_VECTOR). Logical operators are

XOR , AND ….etc.

Ex: y<= a XOR b;

c) Arithmetic operators:

'+' , '-' , '*' ……etc are the Arithmetic operators

If the data is of type integer, signed or unsigned arithmetic operations directly can be performed on them. But these are not allowed if data is of type std_logic_vector.

Eg: signal x,y,z:signed(11 downto 0);

z<=x+y.

d) concatenation operators:

they are used to group values. The data can be of any of the types listed for logical operations.

The concatenation operators are

& , (, , , ,) etc

Examples:

X<= ‘1’

Y<= x & “100001” then y <= “1100001”

Z<= (‘1’ , ‘1’ , ‘0’ , ‘0’ , ‘0’ , ‘0’ , ‘1’) then z <= “1100001”

Attributes :

The purpose of attributes is to give VHDL more flexibility, and also allow the construction of generic pieces of code(code that will work for any vector or array size). They are 2 types.

Data attributes: return information(a value) regarding a data vector.

Signal attributes: serve to monitor a signal (return TRUE or FALSE).

Data attributes: predefined synthesizable data attributes are following:

D’low : returns lower array index.

D’high : returns higher array index.

D’range : returns vector range.

Ex: vector2 is array (0 to 31) of STD_LOGIC_VECTOR (11 downto 0));

Signal x : vector2;

For i in x’range loop ------x’range = 0 to 31

End loop;

Signal attributes: predefined mostly used synthesizable Signal attribute is following:

S’event: returns true when an event(change) occurs on signal ‘s’.

Ex: if clk’event and clk =’1’ then

End if;

Here if statement will be executed only when clk signal changes from 0 to 1.

Sequential code:

Processes , functions , procedures are the only sections of code that are executed sequentially.

The statements used in this section are all sequential.they are IF , WAIT, LOOP and CASE.

Which are allowed only in PROCESS,FUNCTION ……etc.

Process: its syntax is shown below

[label:] Process(sensitivity list)

[variable name type [range] [:= initial_value;]]

Begin

(sequential code)

End process [label];

If variables used they must be declared in declararive part of process (before the word **begin**) sensitivity list describes the parameters which are changing inside the process.

Signals and variables:

Signals can be declared in a package, entity or architecture(in declarative part) . it is global and can be used any where in architecture code.

Variable are declared only in sequential code (eg: in process). it is local and It should used only inside of process.

IF : it is used only inside a function,procedure or process

Syntax:

If condition then assignments;

Elsif conditions then assignments;

……..

Else assignments;

End if;

Example:

If x=”1000” then Temp<=”10011000”;

elsIf x=”1001” then Temp<=”11111000”;

else Temp<=”11001000”;

End if;

Loop: it is useful when a piece of code must be instantiated several times.like if, wait and case, loop is intended exclusively for sequential code ie inside a function,procedure or process. there are several ways of using loop (eg for and while). For loop is mostly preferred in vhdl code.

Syntax:

[label:] for identifier in range loop

(sequential statements)

End loop [label];

Example:

For I in o to 5 loop

y(i) <= x(i+2) XOR en ;

end loop;

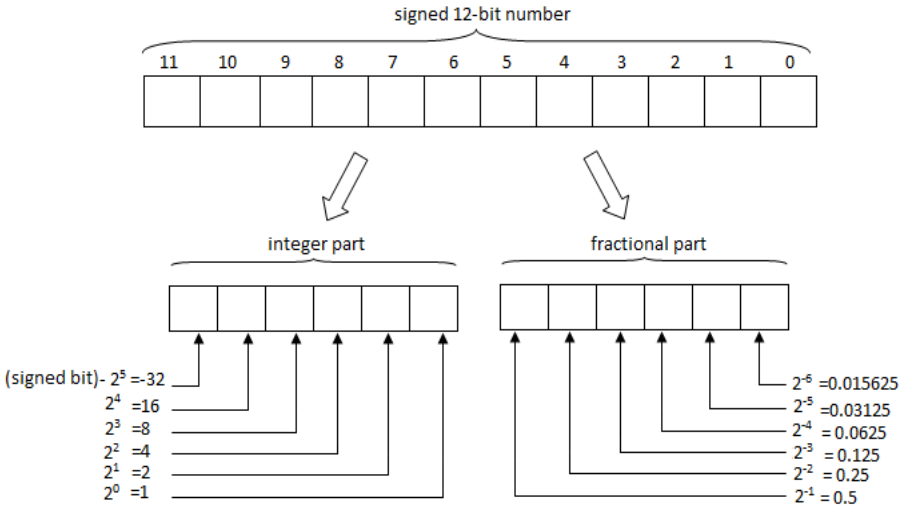
case : it is also intended exclusively for sequential code.

Example :

```
Case sel is
        When “00”  =>  x <= a;
        When “01”  =>  x <= b;
        When “10”  =>  x <= c;
       When others =>  x<=d;
End case;
```

Signed binary fixed point representation:

Signed binary fixed point representation is necessary since IFFT output contains mostly the decimal values.



12-bit signed binary 6.6 fixed point representation

Eg: if x(11:0) = “111111100000” is taken. Actually it represents ‘-32’

After 6.6 precision is consider x(11:0) =“111111.100000 “ it represents ‘-0.5’ .

Barrel shifter:

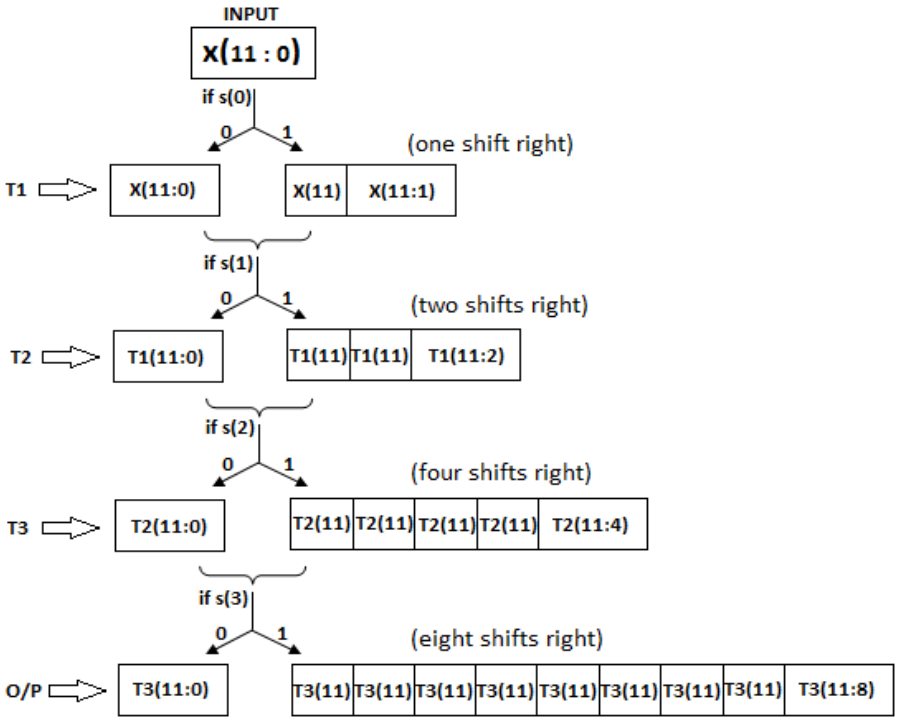
A barrel shifter which can shift 15 positions right is shown in fig. here Barrel shifter used to perform division operation by shifting right to required positions. Here input is 12-bit signed vector. Amount of shift is specified by shift input s(3 : 0).

Eg: If s(3:0) = “1000” then it shifts the input by 8 positions to the right.

signed numbers most significant bit(MSB) represents sign bit. Since barrel shifter shifts input of signed data type after right shift the emptied left most position always should be replaced by sign bit (i.e MSB).

Suppose if x(11:0) =“1101 0000 0000” (-12) and shift input s(3:0)= “1000” after 8 shifts to right x(11:0) = “1111 1111 1101” (-0.046875).

Barrel shifter diagram:



FSM(finite state machines):

it is special modeling technique for sequential logic circuits. This model can be very helpful in the design of certain types of systems, particularly those whose tasks form a well-defined sequence(eg digital controllers). FSM contains two sections.

Lower section contains the sequential logic(flip-flops), while the upper section contains combinational logic because in this section no signal assignment is made at the transition of another signal .

The combinational (upper) section has two inputs, one being pr_state(present state) and the other the external input proper. It has two outputs, nx_state (next state) and the external output proper. However sequential code can implement both types of logic, combinational as well as sequential. So upper part can also be implemented using process.

design of upper (combinational) section

```
process(input, pr_state)
    begin
       case pr_state is
         when state0 =>
           if ( input = …)  then
              output <=< value>;
              nx_state < state1;
           else …
           end if;
         when state1 =>
           if ( input = …)  then
              output <=< value>;
              nx_state < state2;
           else …
           end if;
end case;
end process;
```

as can be seen this code does two things:(a) it assigns the output value and (b) it establishes the next state. Here design of combinational circuits using sequential statements for all input signals are present in the sensitivity list and all input/output combinations are specified.

The sequential(lower) section has three inputs (clock,reset, and nx_state) and one output(pr_state). Since all flip-flops are in this part of system clock and reset must be connected to it.

Design of lower section

```
Process(reset, clock)
Begin
  If(reser ='1') then
     pr_state <= state_0;
 elsif(clock'event and clock ='1')  then
     pr_state <= nx_state;
end if;
end process;
```

this code consists of an asynchronous reset, which determines the initial state of the system(state0), followed by the synchronous storage of nx_state(at the positive transition of clock), which will produce pr_state at the lower section's output. Another advantage of this design style is that the number of registers is minimum. The number of flipflips inferred from the code is simply equal to the number of bits needed to encode all states of the FSM(because the only signal to which a value is assigned at the transition of another signal is pr_state). Hence if default encoding style is used, just [$\log_2 n$] flipflops will then be needed, where n is the number of states.

Chapter 3

implementation of individual blocks in MSE-OFDM including random binary number generator , QAM and IFFT techniques

binary source:

In Tx block binary source is replaced by Random binary number generator . random Random binary number generators can automatically create long runs of numbers with good random properties but eventually the sequence repeats. Here Random binary number generator uses linear feedback shift register

linear feedback shift register(LFSR): a linear feedback shift register is a shift register whose input bit a linear function of its previous state. The only linear function of single bits is XOR, thus it is a shift register whose input bit is driven by the exclusive-or of some bits of the over all shift register value. A shift registers can shift the data either left or right . The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current or previous state. the register has a finite number of possible states, it must eventually enter a repeating cycle. Generally n-bit shift register repeat itself after "2^n -1" However, an LFSR with a wellchosen feedback function can produce a sequence of bits which appears random and which has a very long cycle. The main terms used are feedback, clock, input, tapping and shift direction

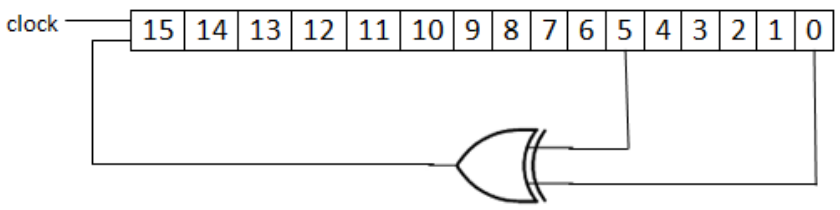
clock: here shift register shift contents only after clock input changes from zero to one.

Feedback: before the shift register shift the contents some bit values are driven by EX-OR and the output is feedback to input.

Tapping: the selected bit positions for feedback function is called tapping.

Shift : here shift direction is from right to left. With each shift LFSR moves to new state.

Input: After shift(left) the emptied position (last bit here) is filled by the feedback output.



A 16 bit LFSR rand_bin(15:0) with feedback is shown in following figure. Here the tap values are taken from bit positions 5,0 respectively. At every clock cycle, 16 bits are used as outputs and "shifted". This technique can be easily coded in VHDL and generates almost no extra FPGA logic cell. The values in register at different clock cycles for a particular seed value is shown in following table. The values 5,0 specifies primitive polynomial x^5+1.

Register states(tap values are bit position 5 and 0 respectively)																	
Bit position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	output
Clock 0 (seed)	1	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0	“1010000011010100”
Clock 1	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	“0100000110101000”
Clock 2	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	“1000001101010001”
Clock 3	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	“0000011010100011”
Clock 4	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	0	“0000110101000110”
Clock 5	0	0	0	1	1	0	1	0	1	0	0	0	1	1	0	0	“0001101010001100”
. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .	. . . . . .

.

QAM(Quadrature Amplitude Modulation) :

Hardware implementation: Quadrature amplitude modulation (QAM) requires changing the phase and amplitude of a carrier sine wave. One of the easiest ways to implement QAM with hardware is to generate and mix two sine waves that are 90 degrees out of phase with one

another. Adjusting only the amplitude of either signal can affect the phase and amplitude of the resulting mixed signal.

These two carrier waves represent the in-phase (I) and quadrature-phase (Q) components of our signal. Individually each of these signals can be represented as:

$I = A\cos(\varphi)$ and $Q = A\sin(\varphi)$.

Note that the I and Q components are represented as cosine and sine because the two signals are 90 degrees out of phase with one another. Using the two identities above and the following trigonometric identity

$\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$,

rewrite a carrier wave $A\cos(2\pi fct + \varphi)$ as

$A\cos(2\pi fct + \varphi) = I\cos(2\pi fct) - Q\sin(2\pi fct)$.

As the equation above illustrates, the resulting identity is a periodic signal whose phase can be adjusted by changing the amplitude of I and Q. Thus, it is possible to perform digital modulation on a carrier signal by adjusting the amplitude of the two mixed signals.

M-QAM Symbol Map:

QAM involves sending digital information by periodically adjusting the phase and amplitude of a sinusoidal electromagnetic wave. Here output binary sequence generated from binary source is mapped as symbols by taking n-bits at a time where $n=\log_2 M$.

4-QAM and 16-QAM are mostly preferred among others because of following.

The advantage of 4-QAM is very less bit error present compared to 16-QAM but only 2 bits can be transmitted per each symbol.

The advantage of 16-QAM is more bits can be transmitted per each symbol with considerable bit error. How ever bit error is more compared to 4-QAM.

The number of bits per symbol can be increased by using more than 16-QAM to transmit more bits at a time. But as the number of bits per symbol increasing simultaneously bit error also increases because one symbol may overlap with another. More over gray code symbol mapping

has been used as it allows only one bit error even if one symbol overlap with adjacent symbol.

4-QAM: here 2 bits mapped at a time so that 4-QAM contains any one of symbols "00" or "01" or "10" or "11". outputs of 4-QAM modulator corresponding to above symbols are

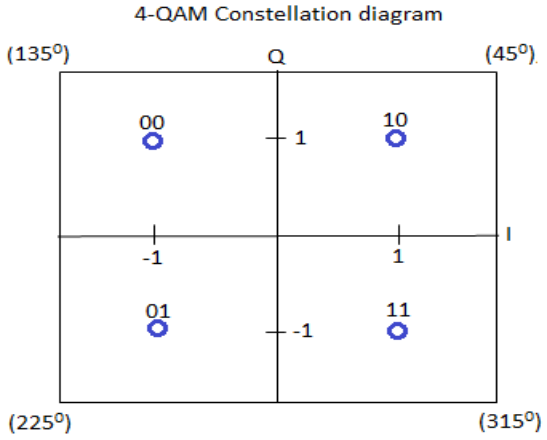
00 -1+1i (111111000000 , 000001000000)

01 -1-1i (111111000000 , 111111000000)

10 1+1i (000001000000 , 000001000000)

11 1-1i (000001000000 , 111111000000)

corresponding constellation diagram to represent unique phase and amplitude for above symbols in 16-QAM is shown below.



Symbol transmitted	Carrier phase (In degrees)	Carrier amplitude
"00"	45	1
"01"	135	1
"10"	225	1
"11"	315	1

Eg:suppose if the generated bit stream is

Rand_bit = (0,1,0,0, 1,0,1,0, 1,0,0,1, 0,0,1,0, 0,1,1,0, 1,1,1,0……..)

So corresponding symbols generated for above sequence for 4-QAM is

Rand_sym =(“01” , “00”, “10” , ”10” , “10” , “01” , “00” , “10” , “01”, “10” , “11” ,10” ……..)

here “111111000000” and “000001000000” are fixed 6.6 representation of -1 and 1 respectively

16-QAM :

If 16-QAM has been used then 4 bits mapped at a time so that 16-QAM contains any of the value among from binary values “0000” to “1111”.

outputs of 4-QAM modulator corresponding to above symbols are

0000 -3+3i (111101000000 , 000011000000)

0001 -3+1i (111101000000 , 000001000000)

0010 -3-1i (111101000000 , 111111000000)

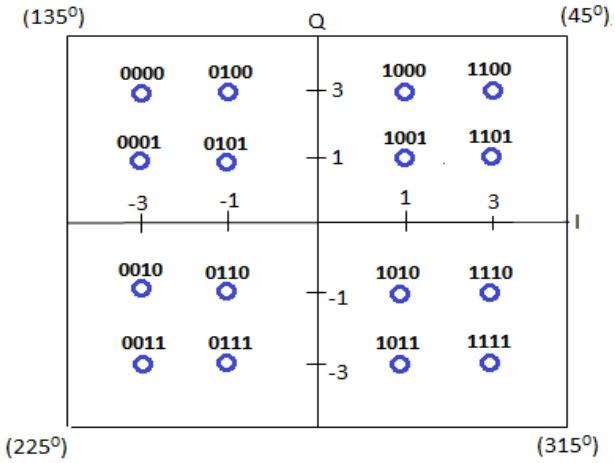
.

.

1111 3-3i (000011000000 , 111101000000)

Since QAM involves sending digital information periodically adjusting the phase and amplitude of each symbol . hence corresponding constellation diagram to represent unique phase and amplitude for above symbols in 16-QAM is shown below.

16-QAM Symbol mapping



Symbol transmitted	Carrier phase (In degrees)	Carrier amplitude
“0000”	135	1
“0001”	165	0.75
“0010”	195	0.75
“0011”	225	1
“0100”	105	0.75
“0101”	135	0.33
“0110”	225	0.33
“0111”	255	0.75
“1000”	75	0.75
“1001”	45	0.33
“1010”	315	0.33
“1011”	285	0.75
“1100”	45	1
“1101”	15	0.75
“1110”	345	0.75
“1111”	315	1

Eg:suppose if the generated bit stream is

Rand_bit = (0,1,0,0, 1,0,1,0, 1,0,0,1, 0,0,1,0, 0,1,1,0, 1,1,1,0…….)

So corresponding symbols generated for above sequence for 16-QAM is

Rand_sym =(“0100”, “1010” “1001” “0010” “0110” “1110” ……..)

IFFT algorithms:

$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, W_N^{-kn}$ where $W_N = e^{\frac{-j2\pi}{N}}$ for k= 0,1,2 ….N-1

Properties of W_N

Periodicity property : W_N^k is periodic with period N . hence $W_N^{k+N} = W_N^k$

Symmetry property: $W_N^{k+\frac{N}{2}} = -W_N^k$ since $e^{\frac{-j2\pi(k+\frac{N}{2})}{N}} = e^{\frac{-j2\pi k}{N}}.e^{-j\pi} = -W_N^k$

Relation: $W_N^k = W_{\frac{N}{k}}$ eg: $W_N^2 = W_{\frac{N}{2}}$

RADIX -2 decimation in time IFFT algorithm:

$x(n) = \sum_{k\ even} X(k)\, W_N^{-kn} + \sum_{k\ odd} X(k)\, W_N^{-kn}$

substitute k=2r for even { X(0),X(2),X(4),X(6) }

k=2r+1 for odd { X(1),X(3),X(5),X(7) }

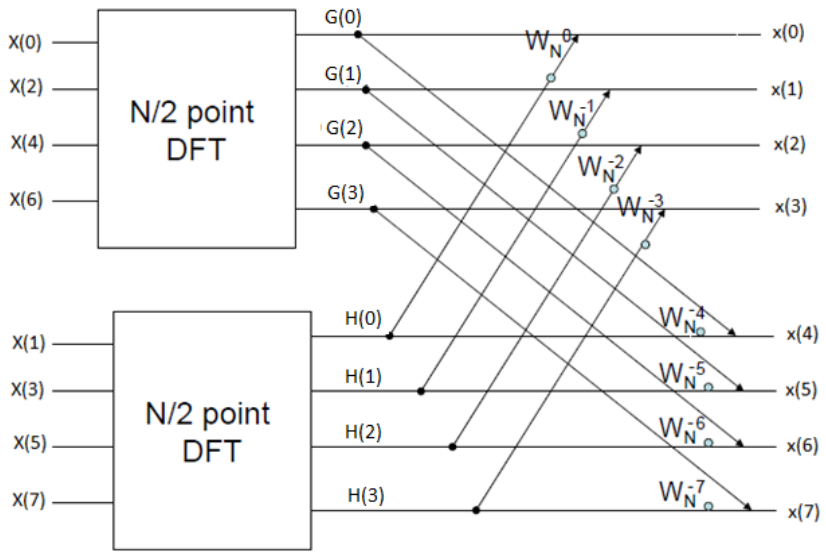
$x(n) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_N^{-2rn} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_N^{-(2r+1)n}$

$x(n) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_{\frac{N}{2}}^{-rn} + W_N^{-n} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_{\frac{N}{2}}^{-rn}$ it can be expressed as

$x(n) = G(n) + W_N^{-n} H(n)$ n=0,1,2 ……. $\frac{N}{2}$ -1 .

here G(n) and H(n) are $\frac{N}{2}$ – point inverse fourier transforms of x(2r) and x(2r+1) respectively.

$x(n+\frac{N}{2}) = G(n) - W_N^{-n} H(n)$ n=0,1,2 ……. $\frac{N}{2}$ -1



G(n) and H(n) can be further decomposed into two smaller inverse fourier transforms and so on.

$$G(n) = \sum_{r=0}^{\frac{N}{2}-1} g(r)\, W_{\frac{N}{2}}^{-rn} = \sum_{l=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{2}}^{-2ln} + \sum_{l=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{2}}^{-(2l+1)n}$$

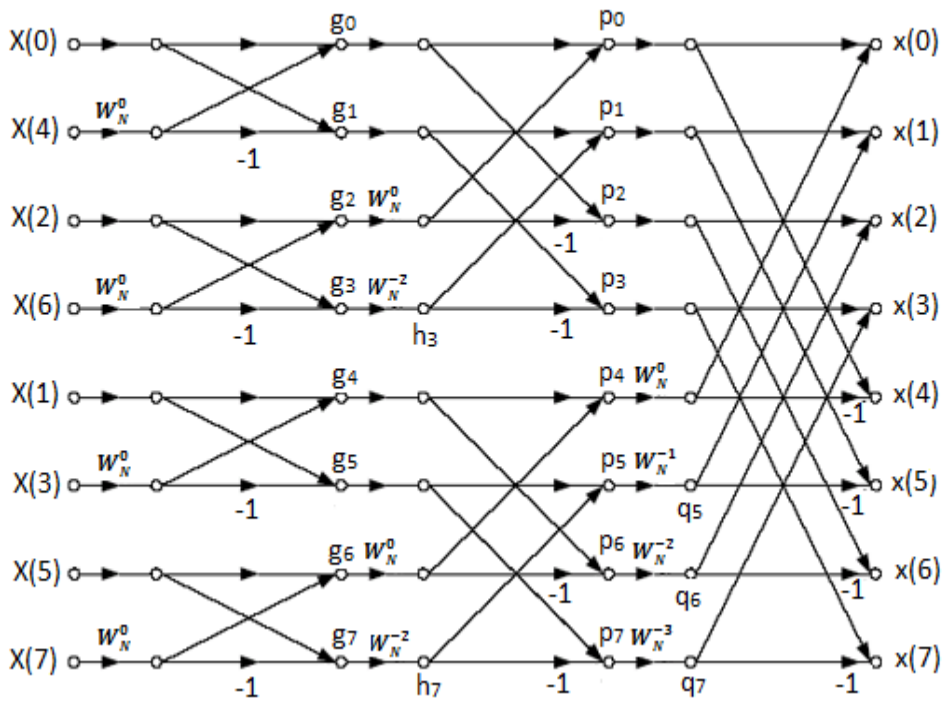
$$= \sum_{l=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{4}}^{-ln} + W_N^{-2n} \sum_{l=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{4}}^{-ln}$$

$$H(n) = \sum_{r=0}^{\frac{N}{2}-1} h(r)\, W_{\frac{N}{2}}^{-rn} = \sum_{l=0}^{\frac{N}{4}-1} h(2l)\, W_{\frac{N}{2}}^{-2ln} + \sum_{l=0}^{\frac{N}{4}-1} h(2l+1)\, W_{\frac{N}{2}}^{-(2l+1)n}$$

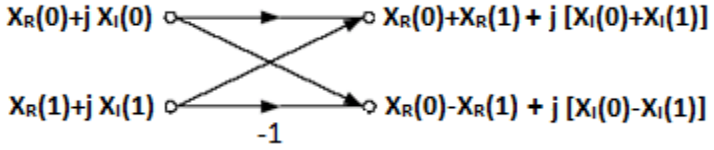
$$= \sum_{l=0}^{\frac{N}{4}-1} h(2l)\, W_{\frac{N}{4}}^{-ln} + W_N^{-2n} \sum_{l=0}^{\frac{N}{4}-1} h(2l+1)\, W_{\frac{N}{4}}^{-ln}$$

Here G(n) and H(n) each represent s the two 2-point IDFT ‘s . so over all four 2-point IDFT are obtained in this stage. Here 8-point IFFT using radix-2 algorithm is shown in figure. From the figure it is shown that after first stage intermediate values are stored in g0,g1,g2……g7 . similarly after second stage intermediate values are stored in p0 , p1, p2…….p7.

8-point IFFT using radix-2 algorithm



Butterfly diagram:



If N=8 has been substituted then values of weights are,

W_N^{-1} = 0.707 + j 0.707 $\qquad$ $W_N^{-2} = j$ $\qquad$ $W_N^{-3} = -0.707 + j\,0.707$

$(P_R + j\,P_I)\ (0.707+j\,0.707) \longrightarrow \underbrace{(P_R - P_I)}_{Q_R}\ (0.707) + j\ \underbrace{(P_R + P_I)}_{Q_I}\ (0.707)$

Binary equivalent of 0.707 and their weight is shown in the following .

0.707 =	1	0	1	1	0	1	0	0	1	1	1	1
	2^{-1}		2^{-3}	2^{-4}		2^{-6}			2^{-9}	2^{-10}	2^{-11}	2^{-12}

$Q_R * (0.707) = \frac{Q_R}{2} + \frac{Q_R}{2^3} + \frac{Q_R}{2^4} + \frac{Q_R}{2^6} + \frac{Q_R}{2^9} + \frac{Q_R}{2^{10}} + \frac{Q_R}{2^{11}} + \frac{Q_R}{2^{12}}$ similarly Q_I

Radix -4 decimation in time IFFT algorithm:

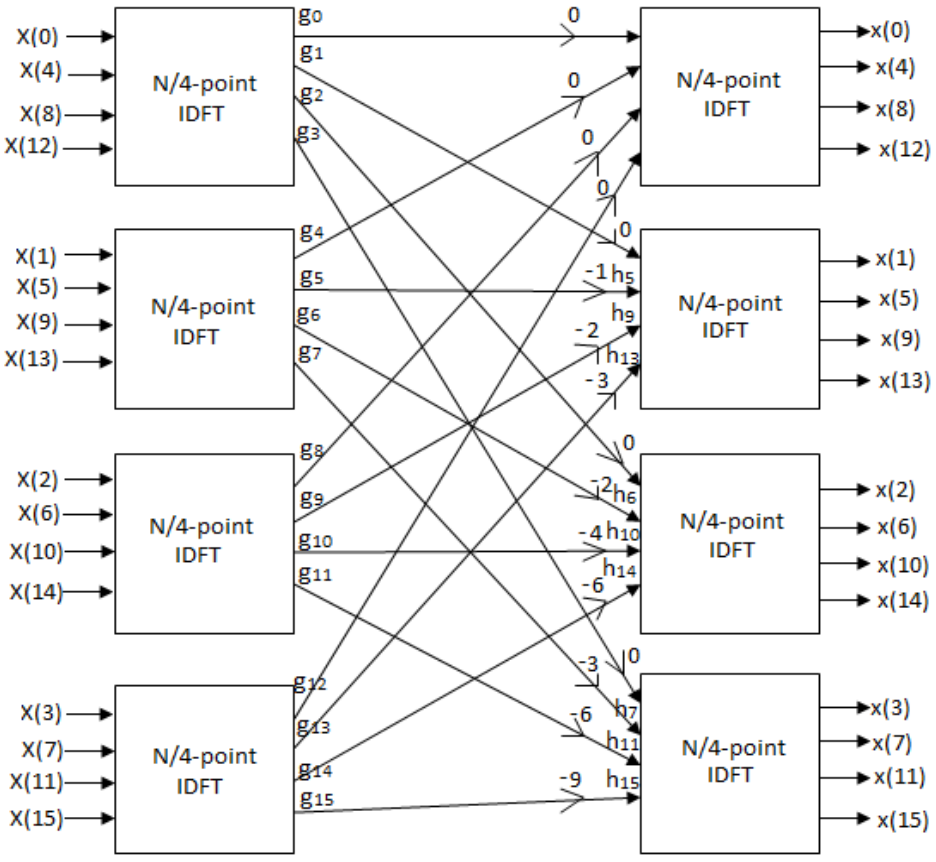
$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, W_N^{-kn}$ where $W_N = e^{\frac{-j2\pi}{N}}$ for k= 0,1,2 ….N-1

$$= \frac{1}{N} \left[\sum_{k=0}^{\frac{N}{4}-1} X(4k)\ W_N^{-4kn} + \sum_{k=0}^{\frac{N}{4}-1} X(4k+1)\ W_N^{-(4k+1)n} + \sum_{k=0}^{\frac{N}{4}-1} X(4k+2)\ W_N^{-(4k+2)n} + \sum_{k=0}^{\frac{N}{4}-1} X(4k+3)\ W_N^{-(4k+3)n} \right]$$

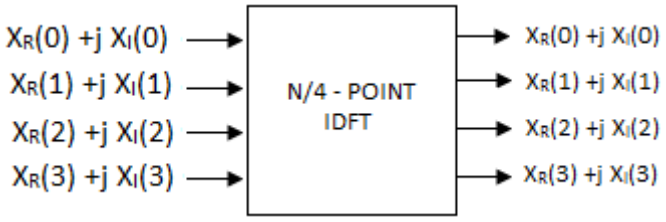
$$= \frac{1}{N} \left[\underbrace{\sum_{k=0}^{\frac{N}{4}-1} X(4k)\, W_{\frac{N}{4}}^{-kn}} + W_N^{-n} \underbrace{\sum_{k=0}^{\frac{N}{4}-1} X(4k+1)\, W_{\frac{N}{4}}^{-kn}} + W_N^{-2n} \underbrace{\sum_{k=0}^{\frac{N}{4}-1} X(4k+2)\, W_{\frac{N}{4}}^{-kn}} + W_N^{-3n} \underbrace{\sum_{k=0}^{\frac{N}{4}-1} X(4k+3)\, W_{\frac{N}{4}}^{-kn}} \right]$$

$$x(n) = \frac{1}{N} \left[G(n) + W_N^{-n}\, H(n) + W_N^{-2n}\, I(n) + W_N^{-3n}\, J(n) \right]$$

16-point IFFT using radix-4 algorithm



4-point IFFT diagram:



where

$x_R(0) = X_R(0) + X_R(1) + X_R(2) + X_R(3)$ $\quad x_I(0) = X_I(0) + X_I(1) + X_I(2) + X_I(3)$

$x_R(1) = X_R(0) - X_R(2) - X_I(1) + X_I(3)$ $\quad x_I(1) = X_I(0) - X_I(2) + X_R(1) - X_R(3)$

$x_R(2) = X_R(0) - X_R(1) + X_R(2) - X_R(3)$ $\quad x_I(2) = X_I(0) - X_I(1) + X_I(2) - X_I(3)$

$x_R(3) = X_R(0) - X_R(2) + X_I(1) - X_I(3)$ $\quad x_I(3) = X_I(0) - X_I(2) - X_R(1) + X_R(3)$

If N=16 has been substituted then values of weights are,

$W_N^{-1} = 0.924 + j\,0.383$ $\quad W_N^{-2} = 0.707 + j\,0.707$ $\quad W_N^{-3} = 0.383 + j0.924$

$W_N^{-4} = j$ $\quad W_N^{-6} = -0.707 + j\,0.707$ $\quad W_N^{-9} = -0.924 - j0.383$

Binary equivalent of 0.707 , 0.383 and 0.924 and their weight are shown in following table.

	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}
0.707 =	1	0	1	1	0	1	0	0	1	1	1	1
0.383 =	0	1	1	0	0	0	1	0	0	0	0	0
0.924 =	1	1	1	0	1	1	0	0	1	0	0	0

Multiplication of these values shown in above table with another values is similar to shown in 8-point IFFT radix-2 algorithm.

Chapter 4

Results and discussions

MSE-OFDM is implemented in following ways.

1. 32- bit LFSR, 4-QAM, two 8 point IFFT
2. 64 - bit LFSR, 16-QAM, two 8 point IFFT
3. 128 - bit LFSR, 16-QAM, four 8 point IFFT
4. 256 - bit LFSR, 16-QAM, four 16 point IFFT

All are synthesized and implemented on virtex-5 xc5vlx30-2ff676 based device. XST synthesis tool has been used to synthesis the model.

1. 32- bit LFSR, 4-QAM, two 8 point IFFT :

Rand_bin (31:0) = "10 10 01 00 11 10 10 10 10 01 00 01 00 10 10 00";

4 - QAM ; IFFT = 8 point IFFT(two blocks) using radix-2

Table for results(QAM output):

S.NO	Bit to symbols	Output of 4-QAM (In MATLAB)	Real output of 4-QAM (in VHDL)	Imaginary output of 4-QAM (in VHDL)
1	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
2	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
3	1	-1-1i	1111 1100 0000(FC0) = -1	1111 1100 0000(FC0) = -1
4	0	-1+1i	1111 1100 0000(FC0) = -1	0000 0100 0000 (040) =1
5	3	1-1i	0000 0100 0000 (040) =1	1111 1100 0000(FC0) = -1
6	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
7	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
8	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
9	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
10	1	-1-1i	1111 1100 0000(FC0) = -1	1111 1100 0000(FC0) = -1

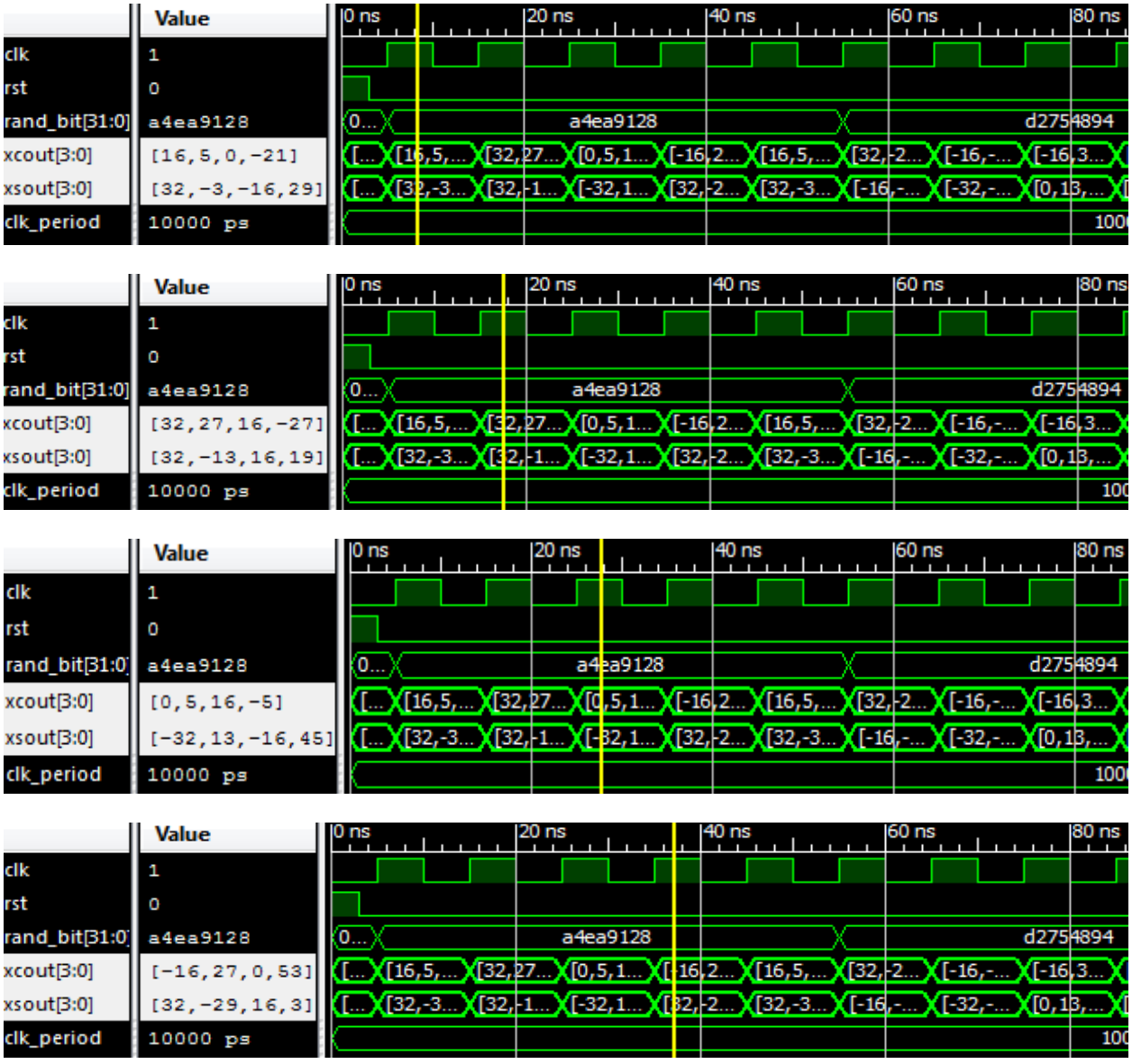
11	0	-1+1i	1111 1100 0000(FC0) = -1	0000 0100 0000 (040) =1
12	1	-1-1i	1111 1100 0000(FC0) = -1	1111 1100 0000(FC0) = -1
13	0	-1+1i	1111 1100 0000(FC0) = -1	0000 0100 0000 (040) =1
14	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
15	2	1+1i	0000 0100 0000 (040) =1	0000 0100 0000 (040) =1
16	0	-1+1i	1111 1100 0000(FC0) = -1	0000 0100 0000 (040) =1

Table for results(IFFT output):

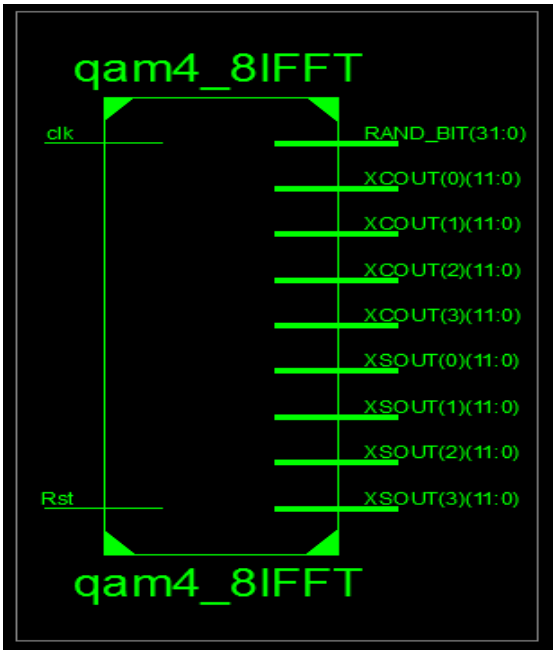
S.NO	Output of IFFT after cyclic prefix (in MATLAB)	Real O/P of IFFT after cyclic prefix (in VHDL)	Imaginary O/P of IFFT after cyclic prefix (in VHDL)
1	0.25+0.5i	010(16) = 0.25	020(32) = 0.5
2	0.0732-0.0732i	005(5) = 0.0781	FFD(-3) = -0.047
3	0-0.25i	000(0) = 0	FF0(-16) = -0.25
4	-0.28+0.426i	FEB(-21) = -0.32	01D(29) = 0.45
5	0.5+0.5i	020(32) = 0.5	020(32) = 0.5
6	0.426-0.176i	01B(27) = 0.421	FF3(-13) = -0.20
7	0.25+0.25i	010(16) = 0.25	010(16) = 0.25
8	-0.426+0.323i	FE5(-27) = -0.42	013(19) = 0.297
9	0-0.5i	000(0) = 0	FE0(-32) = -0.5
10	0.0732+0.176i	005(5) = 0.078	00D(13) = 0.20
11	0.25-0.25i	010(16) = 0.25	FF0(-16) = -0.25
12	-0.0732+0.676i	FFB(-5) = -0.078	02D(45) = 0.703
13	-0.25+0.5i	FF0(-16) = -0.25	020(32) = 0.5
14	0.426-0.426i	01B(27) = 0.422	FE3(-29) = -0.45
15	0+0.25i	000(0) = 0	010(16) = 0.25

16	0.78+0.0732i	035(53) = 0.828	003(3) = 0.047
17	0.25+0.5i	010(16) = 0.25	020(32) = 0.5
18	0.0732-0.0732i	005(5) = 0.0781	FFD(-3) = -0.047
19	0-0.25i	000(0) = 0	FF0(-16) = -0.25
20	-0.28+0.426i	FEB(-21) = -0.32	01D(29) = 0.45

Simulation results:



RTL Schematic



Area factor:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	87	19200	0%
Number of Slice LUTs	1235	19200	6%
Number of fully used LUT-FF pairs	29	1293	2%
Number of bonded IOBs	130	400	32%
Number of BUFG/BUFGCTRLs	1	32	3%

Total adders/subtractors used here are 192.

Speed factor:

Maximum frequency of operation = 643.430 Mhz.

power factor:

Clock Frequency(Mhz)	Total quiscent power(watts)	Total dynamic power(watts)	Total power(watts)
10	0.378	0.008	0.386
20	0.378	0.014	0.392
50	0.378	0.033	0.410
100	0.378	0.063	0.441

2.Rand_bin = 64 bits ; 16-QAM ; IFFT = 8 point IFFT (radix-2)

Table for results(QAM output):

S.NO	Bit to symbols	Output of 16-QAM (In MATLAB)	Real output of 16-QAM (in VHDL)	Imaginary output of 16-QAM (in VHDL)
1	8 (1000)	1+3i	0000 0100 0000(040) = 1	0000 1100 0000(0C0) = 3
2	4 (0100)	-1+3i	1111 1100 0000(FC0) = -1	0000 1100 0000(0C0) = 3
3	4 (0100)	-1+3i	1111 1100 0000(FC0) = -1	0000 1100 0000(0C0) = 3
4	0 (0000)	-3+3i	1111 0100 0000(F40) = -3	0000 1100 0000(0C0) = 3
5	8 (1000)	1+3i	0000 0100 0000(040) = 1	0000 1100 0000(0C0) = 3
6	0 (0000)	-3+3i	1111 0100 0000(F40) = -3	0000 1100 0000(0C0) = 3
7	F (1111)	3-3i	0000 1100 0000 (0C0) = 3	1111 0100 0000(F40) = -3
8	C (1100)	3+3i	0000 1100 0000(0C0) = 3	0000 1100 0000(0C0) = 3
9	2 (0010)	-3-1i	1111 0100 0000(F40) = -3	1111 1100 0000(FC0) = -1

10	4 (0100)	-1+3i	1111 1100 0000(FC0) = -1	0000 1100 0000(0C0) = 3
11	E (1110)	3-1i	0000 1100 0000(0C0) = 3	1111 1100 0000(FC0) = -1
12	A (1010)	1-1i	0000 0100 0000(040) = 1	1111 1100 0000(FC0) = -1
13	9 (1001)	1+1i	0000 0100 0000(040) = 1	0000 0100 0000(040) = 1
14	1 (0001)	-3+1i	1111 0100 0000(F40) = -3	0000 0100 0000(040) = 1
15	2 (0010)	-3-1i	1111 0100 0000(F40) = -3	1111 1100 0000(FC0) = - 1
16	8 (1000)	1+3i	0000 0100 0000(040) = 1	0000 1100 0000(0C0) = 3

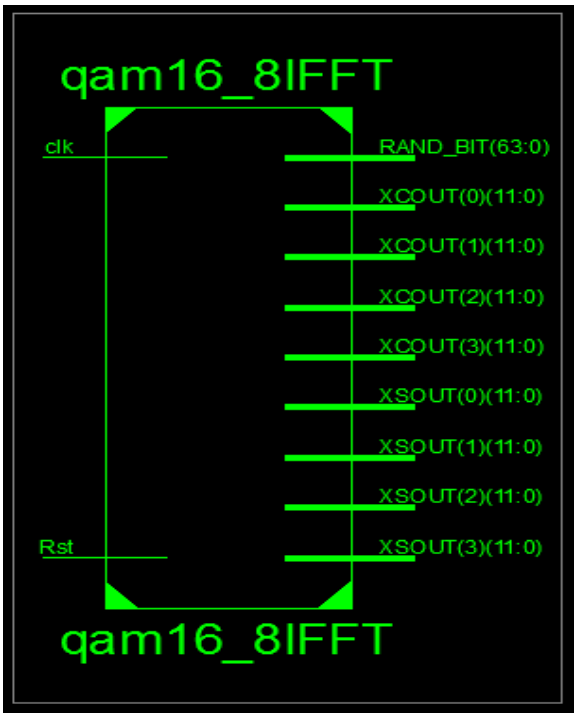
Table for results(IFFT output):

S.NO	Output of IFFT after cyclic prefix (in MATLAB)	Real O/P of IFFT after cyclic prefix (in VHDL)	Imaginary O/P of IFFT after cyclic prefix (in VHDL)
1	0 - 1i	000 (0) = 0	FC0(-64) = -1
2	-0853 - 0.207i	FC8 (-56) = -0.875	FF4 (-12) = -0.187
3	0 + 1i	000 (0) = 0	040 (64) = 1
4	-0.5 - 0.646i	FDE (-34) = -0.531	FD6 (-42) = -0.656
5	0 + 2.25i	000 (0) = 0	090 (144) = 2.25
6	-0.043 - 0.853i	FFD (-3) = -0.046	FC8 (-56) = -0.875
7	0 + 0.25i	000 (0) = 0	010 (16) = 0.25
8	0.0429 + 0.146	003 (3) = 0.046	008 (8) = 0.125
9	1 - 0.75i	040 (64) = 1	FD0 (-48) = -0.75
10	-1.457 - 0.146i	FA3 (-94) = -1.453	FF8 (-8) = -0.125
11	0 + 1.25i	000 (0) = 0	050 (80) = 1.25
12	1.457 + 0.853i	05D (93) = 1.453	038 (56) = 0.875
13	-0.5 + 0.5i	FE0 (-32) = -0.5	020 (32)= 0.5
14	-0.146 + 1.207i	FF8 (-8) = -0.125	04C (76) = 1.187
15	-0.5 - 0.5i	FE0 (-32) = -0.5	FE0 (-32)= -0.5
16	-0.5 - 1.353I	FE2(-30) = -0.468	FAA (-86)= -1.343
17	0 - 1i	000 (0) = 0	FC0 (-64)= -1

18	-0853 - 0.207i	FC8 (-56) = -0.875	FF4 (-12)= -0.187
19	0 + 1i	000 (0) = 0	040 (64)= 1
20	-0.5 - 0.646i	FDE (-34) = -0.531	FD6 (-42)= -0.656

Simulation results:

RTL Schematic



Area factor:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	166	19200	0%
Number of Slice LUTs	1250	19200	6%
Number of fully used LUT-FF pairs	67	1349	4%
Number of bonded IOBs	162	400	40%
Number of BUFG/BUFGCTRLs	1	32	3%

Total adders/subtractors used here are 192 .

Speed factor:

Maximum frequency of operation = 640.38 Mhz.

power factor:

Clock Frequency(Mhz)	Total quiscent power(watts)	Total dynamic power(watts)	Total power(watts)
10	0.378	0.011	0.389
20	0.378	0.018	0.396
50	0.378	0.040	0.418
100	0.378	0.075	0.453

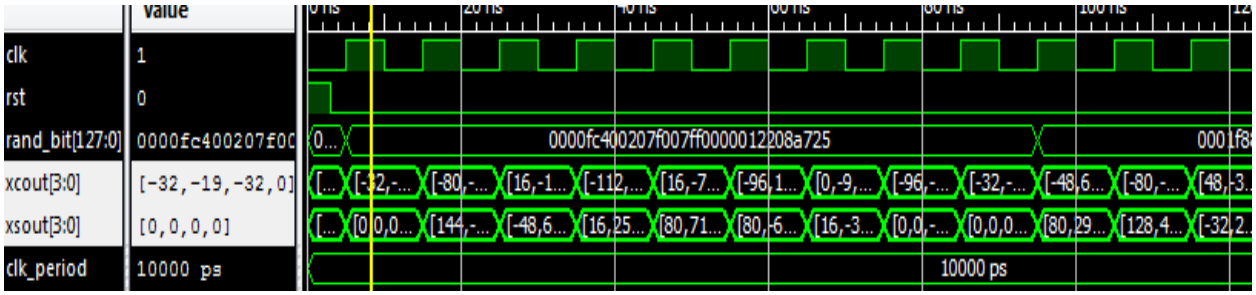
3.For rand_bin= 128 bits; 16-QAM; 8 point IFFT(four blocks) using radix-2

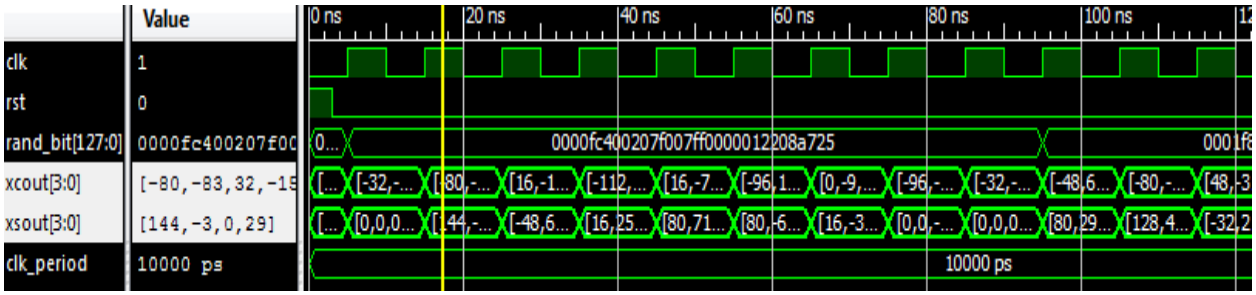
Table for results(IFFT output):

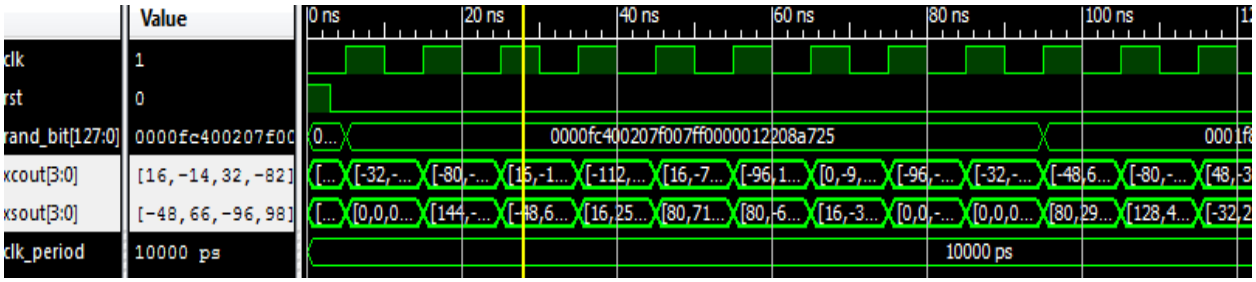
S.NO	Output of IFFT after cyclic prefix (in MATLAB)	Real O/P of IFFT after cyclic prefix (in VHDL)	Imaginary O/P of IFFT after cyclic prefix (in VHDL)
1	-0.5	FE0 (-32)= -0.5	000 (0) =0
2	-0.29	FED (-19)= -0.296	000 (0) =0
3	-0.5	FE0 (-32)= -0.5	000 (0) =0
4	0	000 (0)= 0	000 (0) =0
5	-1.25+2.25i	FB0(-80)= -1.25	090(144) =2.25
6	-1.28-0.03i	FAD(-83) =-1.296	FFD(-3) = -0.0468
7	0.5	020(32)= 0.5	000(0) = 0
8	-0.22+0.47i	FF1(-15)=-0.234	01D(29) = 0.453
9	0.25-0.75i	010(16)=0.25	FD0(-48) = -0.75
10	-0.219+1.03i	FF2(-14)=-0.218	042(66) = 1.031
11	0.5-1.5i	020(32) = 0.5	FA0(-96) = -1.5
12	-1.28+1.53i	FAE(-82) = -1.28	062(98) = 1.531

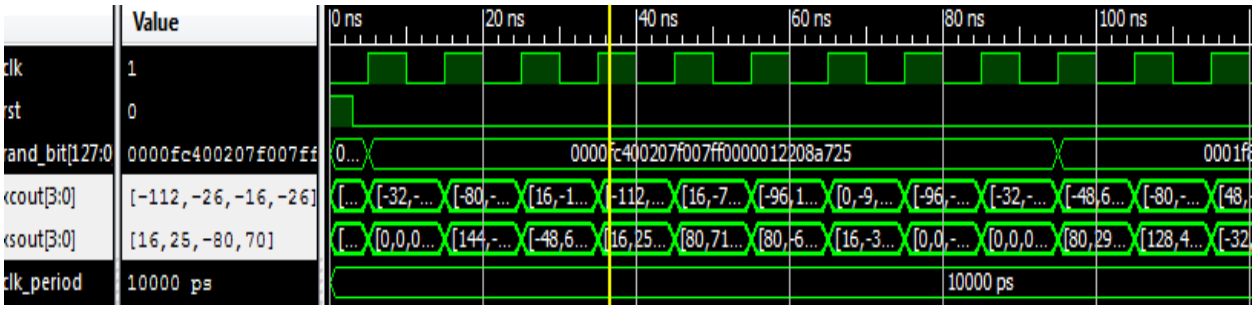
13	-1.75+0.25i	F90(-112) = -1.75	010(16) =0.25
14	-0.396+0.396i	FE6(-26)= -0.406	019(25) = 0.39
15	-0.25-1.25i	FF0(-16)=-0.25	FB0(-80) = -1.25
16	-0.396+1.103i	FE6(-26) = -0.406	046(70) = 1.093
17	0.25+1.25i	010(16) = 0.25	050(80) = 1.25
18	-1.103+1.103i	FB9(-71)= -1.109	047(71) = 1.109
19	1.75-0.25i	070(112)= 1.75	FF0(-16)= -.25
20	-1.103+0.396i	FB9(-71)= -1.109	019(25) = 0.3906
21	-1.5+1.25i	FA0(-96) = -1.5	050(80) = 1.25
22	1.633-0.926i	068(104) = 1.625	FC4(-60) = -0.937
23	1.25	050(80) = 1.25	000(0) = 0
24	0.57+0.487i	024(36)= 0.562	01E(30) = 0.468
25	0.25i	000(0) = 0	010(16) = 0.25
26	-0.133-0.573i	FF7(-9) = -0.14	FDB(-37) = -0.578
27	0.25-1.5i	010(16) =0.25	FA0(-96) = -1.5
28	0.92-1.98i	03B(59) = 0.921	F81(-127) = -1.984
29	-1.5	FA0 (-96) = -1.5	000 (0) = 0
30	-1.707i	F92 (-110) = -1.718	000 (0) = 0
31	1.5-1i	060 (96) = 1.5	FC0 (-64) = -1
32	0	000 (0) =0	000 (0) = 0
33	-0.5	FE0 (-32) = -0.5	000 (0) = 0
34	-0.29	FED (-19) = -0.296	000 (0) = 0
35	-0.5	FE0 (-32) = -0.5	000 (0) = 0
36	0	000 (0) =0	000 (0)= 0

Simulation results

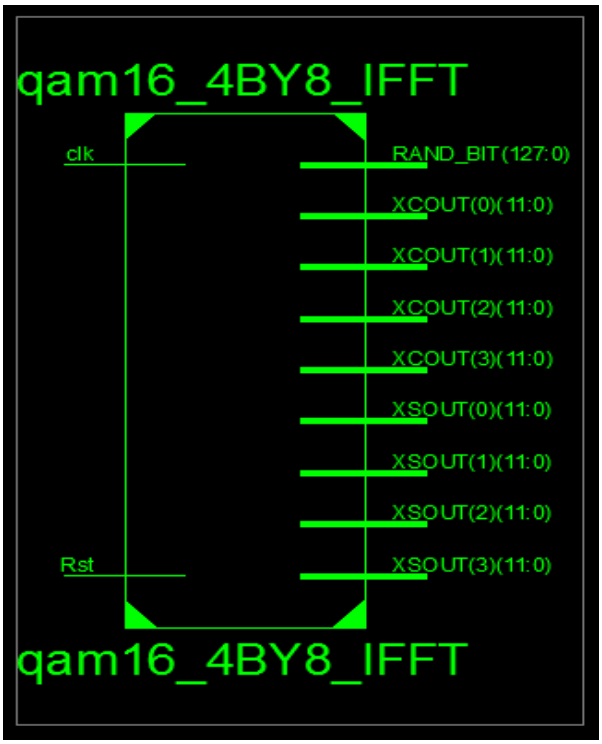








RTL schematic:



Area factor:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	330	19200	1%
Number of Slice LUTs	2333	19200	12%
Number of fully used LUT-FF pairs	125	2538	4%
Number of bonded IOBs	226	400	56%
Number of BUFG/BUFGCTRLs	1	32	3%

Total adders/subtractors used here are 336 .

Speed factor:

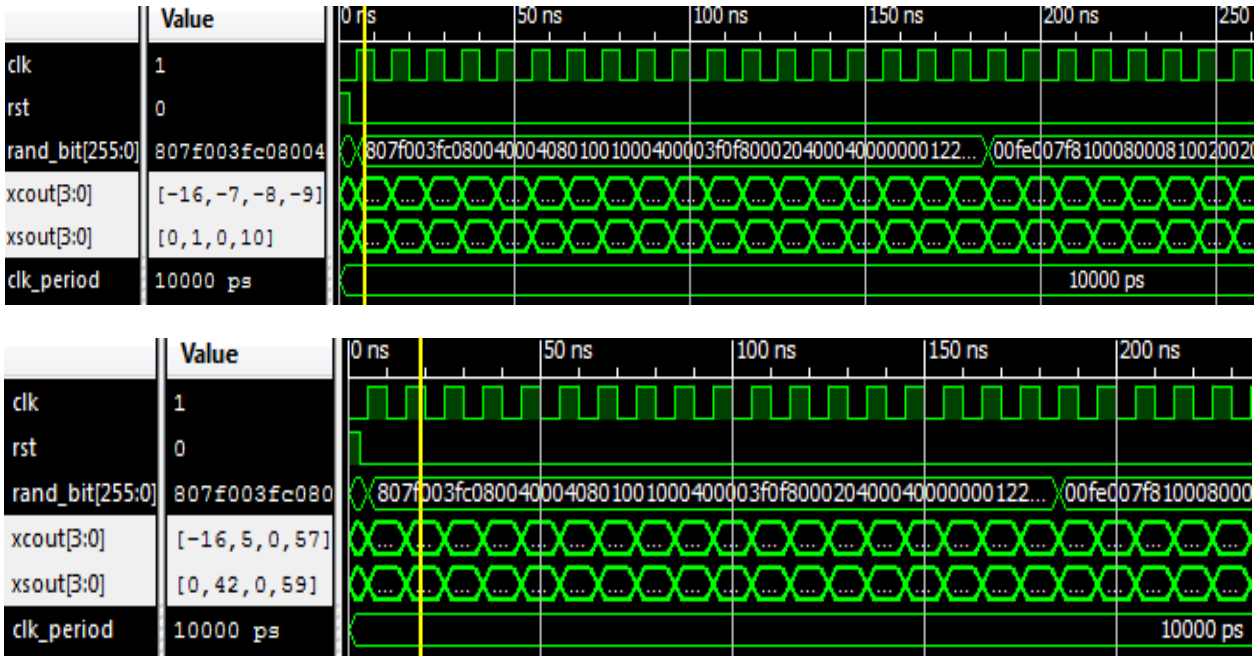
Maximum frequency of operation = 632.132 Mhz.

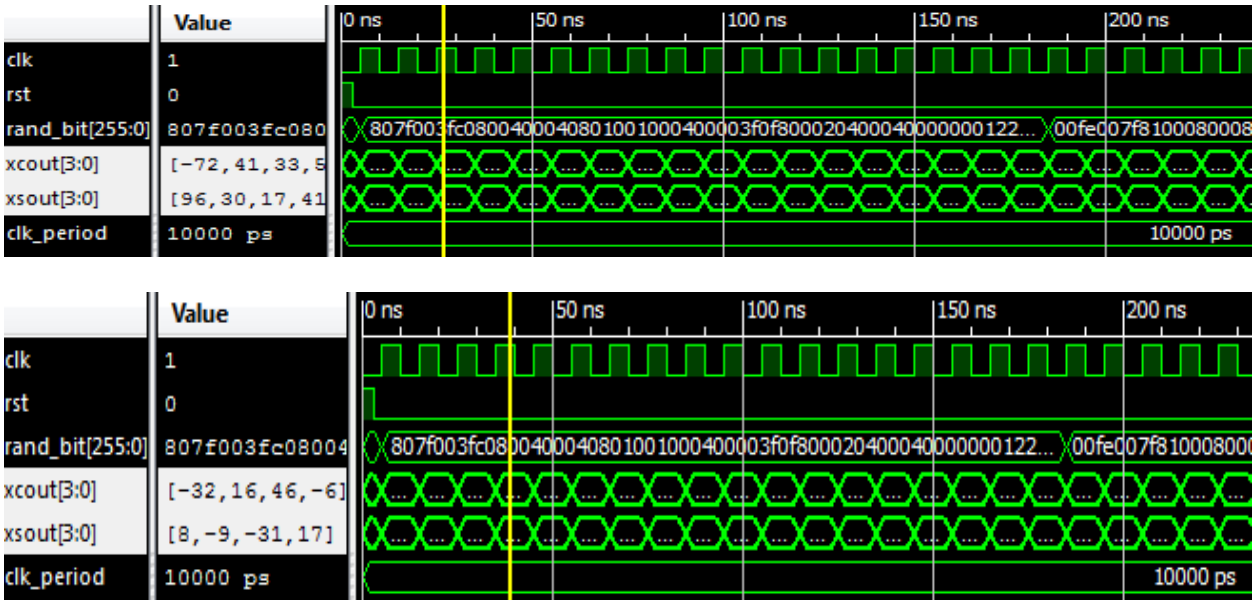
Power factor:

Clock Frequency(Mhz)	Total quiscent power(watts)	Total dynamic power(watts)	Total power(watts)
10	0.378	0.02	0.398
20	0.378	0.023	0.401
50	0.378	0.035	0.413
100	0.378	0.053	0.431

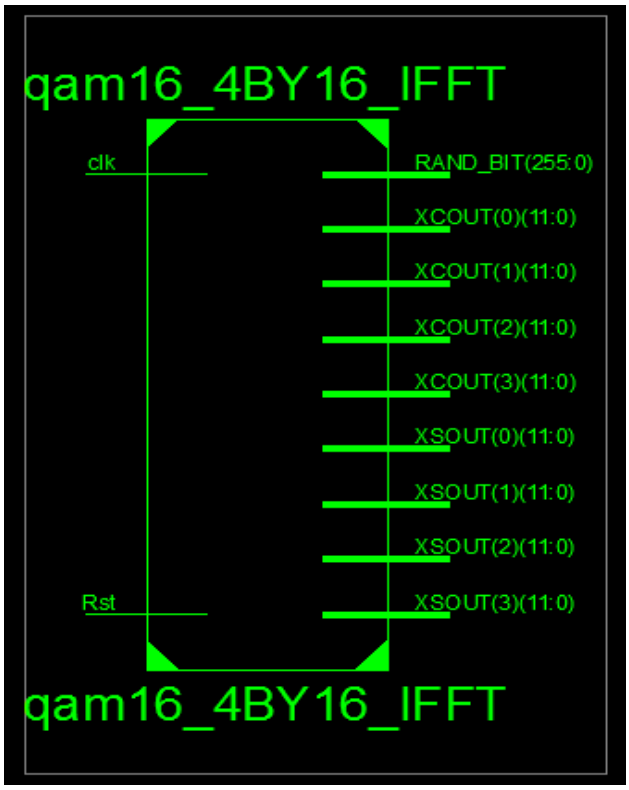
4.For rand_bin= 256 bits; 16-QAM; 16 point IFFT(four blocks) using radix-4

Simulation results





RTL Schematic



Area factor:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	660	19200	3%
Number of Slice LUTs	8988	19200	46%
Number of fully used LUT-FF pairs	254	9394	2%
Number of bonded IOBs	354	400	88%
Number of BUFG/BUFGCTRLs	1	32	3%

Total adders/subtractors used here are 844 .

Speed factor:

Maximum frequency of operation = 622.245 Mhz.

power factor:

Clock Frequency(Mhz)	Total quiscent power(watts)	Total dynamic power(watts)	Total power(watts)
10	0.378	0.02	0.398
20	0.378	0.023	0.401
50	0.378	0.035	0.413
100	0.378	0.053	0.431

conclusions

The proposed design is implemented on virtex-5 xc5vlx30-2ff676 based device. XST synthesis tool has been used to synthesis the model. ISE simulator (Isim) has been used for simulation purpose.

The important block in MSE-OFDM is IDFT is implemented using NEDA , a ROM less and multiplier less method.

Since IDFT is implemented using different IFFT algorithms which uses only adders/subtractors and shifters only. The proposed design is hardware efficient compared to other traditional methods that are built using only DA.

Here 12- bit signed fixed point representation is described to get more accurate results for decimal values.

The maximum frequency of operation of proposed design on the mapped FPGA is range of 600 - 700 MHz.

References

[1] Yang Song, Songlin Son, Xiaojun Jing, Hai Huang, "Orthogonality Analysis and Improvement of MSE-OFDM System" , IEEE International Conference on Information, Networking and Automation, pp. 433-437, 2010.

[2] Xianbin Wang, Yiyan Wu and J.-Y. Chouinard, "On the comparison between conventional OFDM and MSE-OFDM systems" IEEE GLOBECOM 2003, vol. 1, pp. 35-39, Dec. 1-5 2003.

[3] S. B. Weinstein and P. M. Ebert, "Data transmission by frequencydivision multiplexing using the discrete Fourier transform," *IEEE Trans. Commun.*, vol. COM-19, no. 5, pp. 628–634, Oct. 1971.

[4] J.Y. Chouinard, X. Wang, and Y. Wu, "MSE-OFDM: A New OFDM Transmission Technique with Improved System Performance", ICASSP 2005, Philadelphia, USA.

[5] Dr Sherif Kishk, Eng Ahmed Mansourf, Prof. Mohy Eldin, "Implementation of an OFDM System using FPGA", IEEE 26th National Radio Science Conference (NRSC), pp. 1-9, March 2009.

[6] Ahmed Saeed, M. Elbably, G.Abdelfadeel, "Efficient FPGA implementation of FFT/IFFT Processor" International Journal of circuits, systems and signal processing, Vol.3, Issue 3, pp. 103-110, 2009.

[7] Zhijian Sun, Xuemei Liu, and Zhongxing Ji, "The Design of Radix-4 FFT by FPGA," International Symposium on Intelligent Information Technology Application Workshops, pp.765-768, 2008.

[8] J. Garcia, R. Cumplido, "On the design of an FPGA-Based OFDM modulator for IEEE 802.11a", 2nd IEEE International Conference on Electrical and Electronics Engineering, pp. 114 – 117, Sept. 2005

[9] Wendi Pan, Ahmed Shams, and Magdy A. Bayoumi, “NEDA: A NEw Distributed Arithmetic Architecture and its Application to One Dimensional Discrete Cosine Transform,” Proc. IEEE Workshop on Signal Processing Syst., pp. 159 – 168, Oct. 1999.

[10] Stanley A. White, “Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review,” IEEE ASSP Magazine, vol. 6, no. 3, pp. 4 – 19, Jul.1989.

[11] F. James, “A Review of Pseudo-random Number Generators, “ Computer physics ommunication 60,1990.

[12] Rajski J, Tyszer J, “On the diagnostic properties of linear feedback shift registers”, ISSN : 0278-0070, IEEE @ 06 August 2002.

[13] Krishnaswamy S, Pillai H K, “On the Number of Linear Feedback Shift Registers With a Special Structure”, ISSN : 0018-9448, IEEE @ 27 February 2012.

[14] Richard M. Jiang, “An Area-Efficient FFT Architecture for OFDM Digital Video broadcasting,” IEEE Trans.Consumer Elect., vol. 53, no. 4, pp. 1322 – 1326, Nov. 2007.

[15] T. Noguchi, Y. Daido, and J. Nossek, “Modulation techniques for microwave digital radio,” ***Commun. Magazine,*** vol. 24, pp. 21-30, Oct. 1986.

[16] Tran-Thong and B. Liu, “Fixed-point fast fourier transform error analysis,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 24, no. 6, pp. 563–573, Dec. 1976.

[17] L. Xiaojin, L. Zongsheng Lai, and C. Jianmin Cui, “A low power and small area FFT processor for OFDM demodulator”, *IEEE Transactions on Consumer Electronics,* 53(2):274-277, May 2007.